

NIM: A Unified Model for Forecasting Weather and Climate

August 17, 2011

Presented by
Jim Rosinski



Developers

- Atmospheric Science
 - Dr. Jin Lee
 - Dr. Alexander MacDonald
 - Dr. Jung-Eun Kim
 - Ka Yee Wong
 - Dr. Ning Wang
 - Dr. Jian-Wen Bao



Developers (cont'd)

- Computer Science
 - Mark Govett
 - Tom Henderson
 - Jacques Middlecoff
 - Paul Madden
 - Jim Rosinski



Atmospheric Science 001

- Forecast vs. Climate
 - Is it going to rain here tomorrow?
 - Will the next 10 years be warmer than the last 10 years?
- Dynamics vs. Physics
 - Dynamics predicts the evolution of the explicitly resolved flow
 - Physics estimates the effects of subgrid-scale processes (e.g. clouds)



“N” in NIM means non-hydrostatic

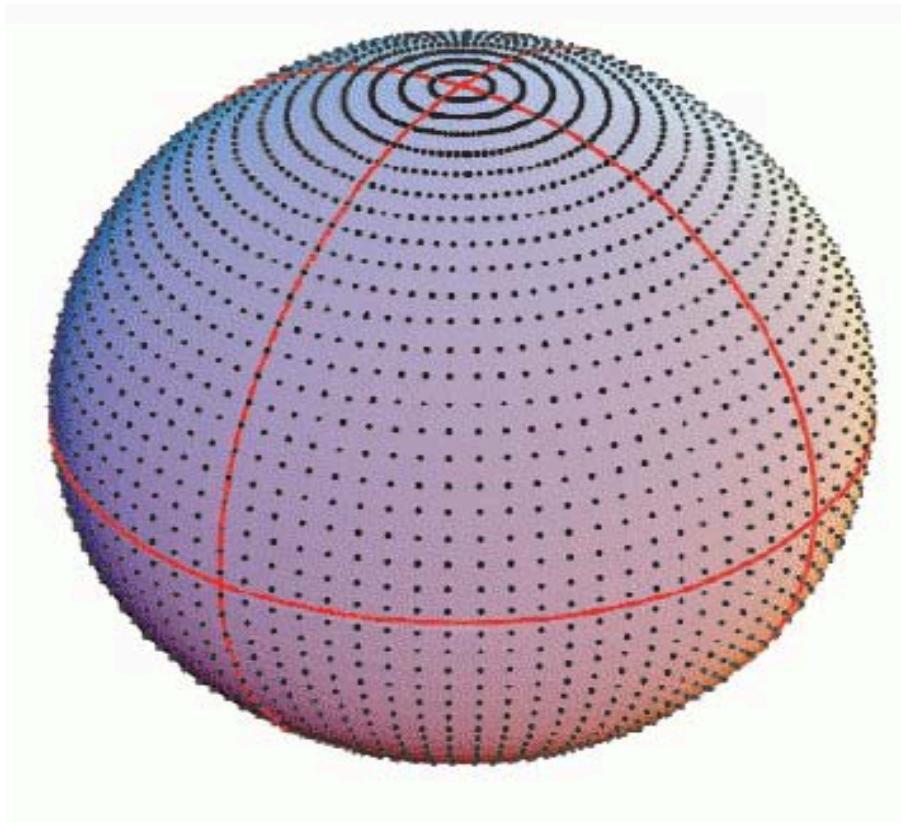
- Hydrostatic equation:

$$\frac{\partial p}{\partial z} = -\rho g$$

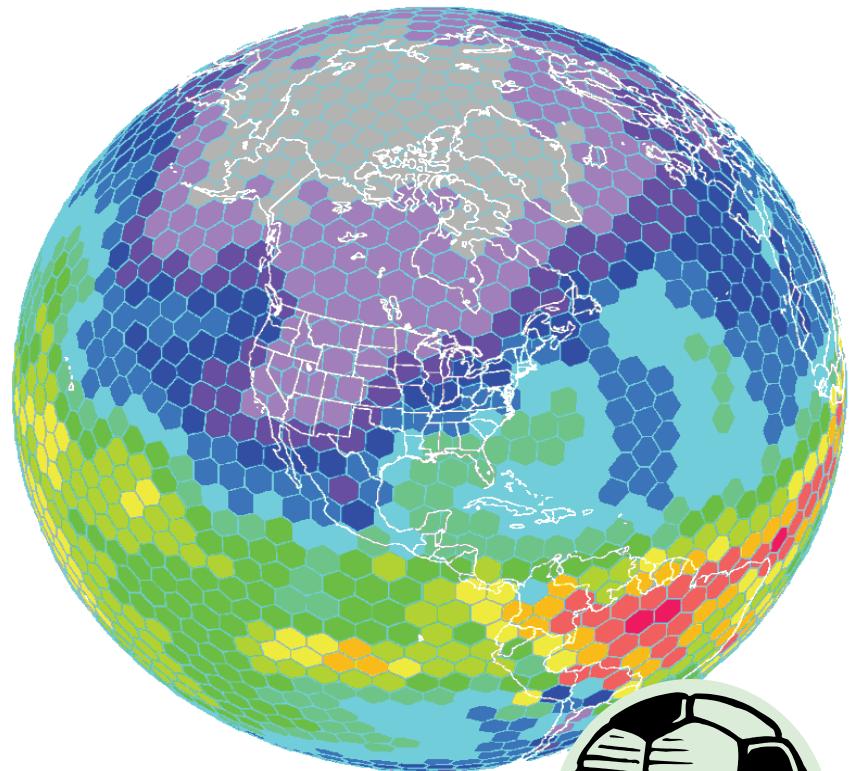


“I” in NIM means Icosahedral

Lat/Lon Model



Icosahedral Model



- Near constant resolution over the globe
- Efficient high resolution simulations

(slide courtesy Dr. Jin Lee)



Resolution nomenclature

!	glvl	Number of grid points	Linear Scale (km)
!	0	12	7,071
!	1	42	3,779
!	2	162	1,174
!	3	642	891
!	4	2562	446
!	5	10,242	223
!	6	40,962	111
!	7	163,842	56
!	8	655,352	28
!	9	2,611,442	14
!	10	10,445,762	7
!	11	41,943,042	3.5
!	12	167,772,162	1.75



Motivation to build NIM

- Unified model useful for both climate and multi-scale forecast studies
- Non-hydrostatic required for high-resolution (<10km)
 - Goal is to run 2KM global forecast model in 2013
- Mass and tracer conservation required for climate
- Computational efficiency required for both forecast and climate
 - Forecast mode requires model to run ~50X realtime
 - Climate simulations require ~1000X realtime



Current Science Status

- Dynamics verified by solutions to synthetic test cases.
 - Analytical wave simulations look good
 - Conserves mass and tracer transports
- GFS, GRIMs dry physics in and running.
- Multi-month simulations completed in aqua-planet mode at G5.

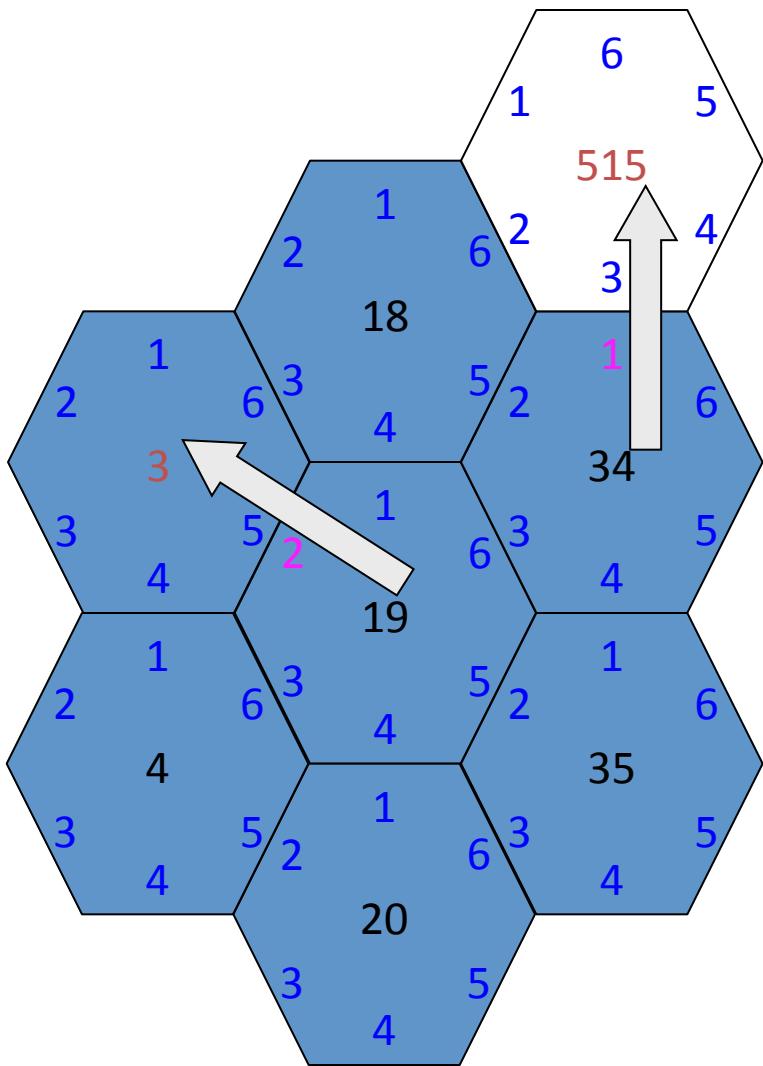


Next Science tasks

- Higher resolution
 - Automate creation of initial and boundary conditions
- Further testing of physics
- Add topography
- Evaluate quality of forecasts
- High resolution equatorial wave simulations



NIM/FIM Indirect Addressing (MacDonald, Middlecoff)



- Single horizontal index
 - Store number of sides (5 or 6) in “nprox” array
 - $\text{nprox}(34) = 6$
 - Store neighbor indices in “prox” array
 - $\text{prox}(1,34) = 515$
 - $\text{prox}(2,19) = 3$
 - Place directly-addressed vertical dimension fastest-varying for speed
 - Very compact code
 - Indirect addressing costs <1%

(slide courtesy Tom Henderson)



Optimize Indirect Addressing

- “Optimize” by reducing access to prox by a factor of nvl
 - Modern compilers do this automatically
 - This is the benefit of inner vertical loop!

```
xnsum = 0.0
do ipn=1,nip          ! Horizontal loop
  do isn=1,nprox(ipn) ! Loop over edges (sides)
    ipp = prox(isn,ipn)
    do k=1,nvl         ! Vertical loop
      xnsum(k,ipn) = xnsum(k,ipn) + x(k,ipp)
    enddo
  enddo
enddo
```



GPU-ization: F2C-ACC, HMPP, PGI

- Desirable characteristics
 - Directive-based (allows single source)
 - Simple syntax (easy to use)
 - If generating CUDA, produce human-readable output (e.g. so user can add printf for debugging)
 - Understood by debuggers
 - Generates portable code (e.g. CUDA or OpenCL)



F2C-ACC: Locally-developed Fortran to CUDA translator

- <http://www.esrl.noaa.gov/gsd/ab/ac/F2C-ACC.html>
- Declare/copy data to the GPU:

```
subroutine copyIn (uw8s)
real, intent (IN) :: uw8s(1:nz,npp,ims:ime,2)

! local variables that will be used on the GPU
real :: vdns(1:nz,npp,ims:ime)

! malloc space on the GPU and copy data to GPU
!ACC$DATA(<uw8s:in>)

! malloc space on the GPU only
!ACC$DATA(<vdns:none>

return
end subroutine copyIn
```



F2C-ACC Translation to CUDA (Input Fortran source)

```
subroutine SaveFlux(nz,ims,ime,ips,ipe,ur,vr,wr,trp,lp,urs,vrs,wrs,trs,rps)
implicit none
<input argument declarations>

!ACC$REGION(<nz>,<ipe-ips+1>,<ur,vr,wr,trp,lp,urs,vrs,wrs,trs,rps:none>) BEGIN
!ACC$DO PARALLEL(1)
do ipn=ips,ipe
!ACC$DO VECTOR(1)
  do k=1,nz
    urs(k,ipn) = ur (k,ipn)
    vrs(k,ipn) = vr (k,ipn)
    trs(k,ipn) = trp(k,ipn)
    rps(k,ipn) = lp (k,ipn)
  end do !k loop
!ACC$THREAD(0)
  wrs(0,ipn) = wr(0,ipn)
!ACC$DO VECTOR(1)
  do k=1,nz
    wrs(k,ipn) = wr(k,ipn)
  end do !k loop
end do !ipn loop
!ACC$REGION END
return
end subroutine SaveFlux
```



Titan Summit



F2C-ACC Translated Code (device)

```
#include "ftnmacros.h"

//!ACC$REGION(<nz>,<ipe-ips+1>,<ur,vr,wr,trp,rp,urs,vrs,wrs,trs,rps:none>) BEGIN
__global__ void saveflux_Kernell(int nz,int ims,int ime,int ips,int ipe,float *ur,float *vr,
                                 float *wr,float *trp,float *rp,float *wrs,float *vrs,float *trs,float *rps) {
    int ipn;
    int k;
//!ACC$DO PARALLEL(1)
    ipn = blockIdx.x+ips;
//    for (ipn=ips;ipn<=ipe;ipn++) {
//!ACC$DO VECTOR(1)
    k = threadIdx.x+1;
//        for (k=1;k<=nz;k++) {
        urs[FTNREF2D(k,ipn,nz,1,ims)] = ur[FTNREF2D(k,ipn,nz,1,ims)];
        vrs[FTNREF2D(k,ipn,nz,1,ims)] = vr[FTNREF2D(k,ipn,nz,1,ims)];
        trs[FTNREF2D(k,ipn,nz,1,ims)] = trp[FTNREF2D(k,ipn,nz,1,ims)];
        rps[FTNREF2D(k,ipn,nz,1,ims)] = rp[FTNREF2D(k,ipn,nz,1,ims)];
//    }
//!ACC$THREAD(0)
    if (threadIdx.x == 0) {
        wrs[FTNREF2D(0,ipn,nz-0+1,0,ims)] = wr[FTNREF2D(0,ipn,nz-0+1,0,ims)];
    }
//!ACC$DO VECTOR(1)
    k = threadIdx.x+1;
//        for (k=1;k<=nz;k++) {
        wrs[FTNREF2D(k,ipn,nz-0+1,0,ims)] = wr[FTNREF2D(k,ipn,nz-0+1,0,ims)];
//    }
//    }
    return;
}
//!ACC$REGION END
```



Titan Summit



F2C-ACC Translated Code (Host)

```
extern "C" void saveflux_ (int *nz__G,int *ims__G,int *ime__G,int *ips__G,int *ipe__G,float *ur,float  
*vr,float *wr,float *trp,float *rp,float *urs,float *vrs,float *wrs,float *trs,float *rps) {  
  
    int nz=*nz__G;  
    int ims=*ims__G;  
    int ime=*ime__G;  
    int ips=*ips__G;  
    int ipe=*ipe__G;  
  
    dim3 cuda_threads1(nz);  
    dim3 cuda_grids1(ipe-ips+1);  
  
    extern float *d_ur;  
    extern float *d_vr;  
< Other declarations>  
  
    saveflux_Kernel1<<< cuda_grids1, cuda_threads1 >>>  
        (nz,ims,ime,ips,ipe,d_ur,d_vr,d_wr,d_trp,d_rp,d_urs,d_vrs,d_wrs,d_trs,d_rps);  
    cudaThreadSynchronize();  
    // check if kernel execution generated an error  
    CUT_CHECK_ERROR("Kernel execution failed");  
    return;  
}
```



Titan Summit



PGI/HPCC Directive Comparison

HMPP

```
!$hmppcg parallel
do ipn=1,nip
  !$hmppcg parallel
    do k=1,nvl
      do isn=1,nprox(ipn)
        xnsnum(k,ipn) = xnsnum(k,ipn) + x(k,ipp)
      enddo
    enddo
  enddo
```

PGI

```
!$acc do parallel
do ipn=1,nip
  !$acc do vector
    do k=1,nvl
      do isn=1,nprox(ipn)
        xnsnum(k,ipn) = xnsnum(k,ipn) + x(k,ipp)
      enddo
    enddo
  enddo
```



(Slide courtesy Tom Henderson)



Current GPU Compiler Limitations

- Limited support for Fortran language features such as modules, derived types
- Both PGI and HMPP prefer “tightly nested outer loops” (not a limitation for F2C-ACC)

```
! This is OK
do ipn=1,nip
    do k=1,nvl
        <statements>
    enddo
enddo
```

```
! This is NOT OK
do ipn=1,nip
    <statements>
    do k=1,nvl
        <statements>
    enddo
enddo
```



(Slide courtesy Tom Henderson)



This is OK with F2C-ACC

```
!ACC$DO PARALLEL(1)
do ipn=ips,ipe
!ACC$DO VECTOR(1)
  do k=1,nz
    fr (k,ipn) = frk*tfr (k,ipn)
    fur(k,ipn) = frk*tfur(k,ipn)
    fvr(k,ipn) = frk*tfvr(k,ipn)
    ftr(k,ipn) = frk*tftr(k,ipn)
  enddo
!ACC$THREAD(0)
  fwr(0,ipn) = frk*tfwr(0,ipn)
!ACC$DO VECTOR(1)
  do k=1,nz
    fwr(k,ipn) = frk*tfwr(k,ipn)
  enddo
enddo
```



Titan Summit



Translated Code

```
//!ACC$DO PARALLEL(1)
    ipn = blockIdx.x+ips;
//    for (ipn=ips;ipn<=ipe;ipn++) {
//!ACC$DO VECTOR(1)
    k = threadIdx.x+1;
//    for (k=1;k<=nz;k++) {
        fr[FTNREF2D(k,ipn,nz,1,ims)] = frk * tfr[FTNREF2D(k,ipn,nz,1,ims)];
        fur[FTNREF2D(k,ipn,nz,1,ims)] = frk * tfur[FTNREF2D(k,ipn,nz,1,ims)];
        fvr[FTNREF2D(k,ipn,nz,1,ims)] = frk * tfvr[FTNREF2D(k,ipn,nz,1,ims)];
        ftr[FTNREF2D(k,ipn,nz,1,ims)] = frk * tftr[FTNREF2D(k,ipn,nz,1,ims)];
//    }
//!ACC$THREAD(0)
if (threadIdx.x == 0) {
    fwr[FTNREF2D(0,ipn,nz-0+1,0,ims)] = frk * tfwr[FTNREF2D(0,ipn,nz-0+1,0,ims)];
}
```



GPU-izing Physics

- Problem 1: Can't thread or block over "k" because most physics has "k" dependence
- Problem 2: In NIM this leaves only 1 dimension for parallelism, and efficient CUDA needs 2



Solution to GPU-izing Physics

- Transpose and “chunking”:

```
!ACC$REGION(<chunksize>,<nchunks>,<dynvars,physvars:none>) BEGIN
    do n=1,nvars_dyn2phy      ! Loop over dyn vars needed in phy
        do k=1,nz              ! Vertical
!ACC$DO PARALLEL (1)
    do c=1,nchunks            ! Chunksize*nchunks >= nip
!ACC$DO VECTOR (1)
    do i=1,chunksize          ! 128 is a good number to choose for chunksize
        ipn = min (ipe, ips + (c-1)*chunksize + (i-1))
        physvars(i,c,k,n) = dynvars(k,ipn,n)
    end do
    end do
    end do
    end do
!ACC$REGION END
```



CPU code runs fast

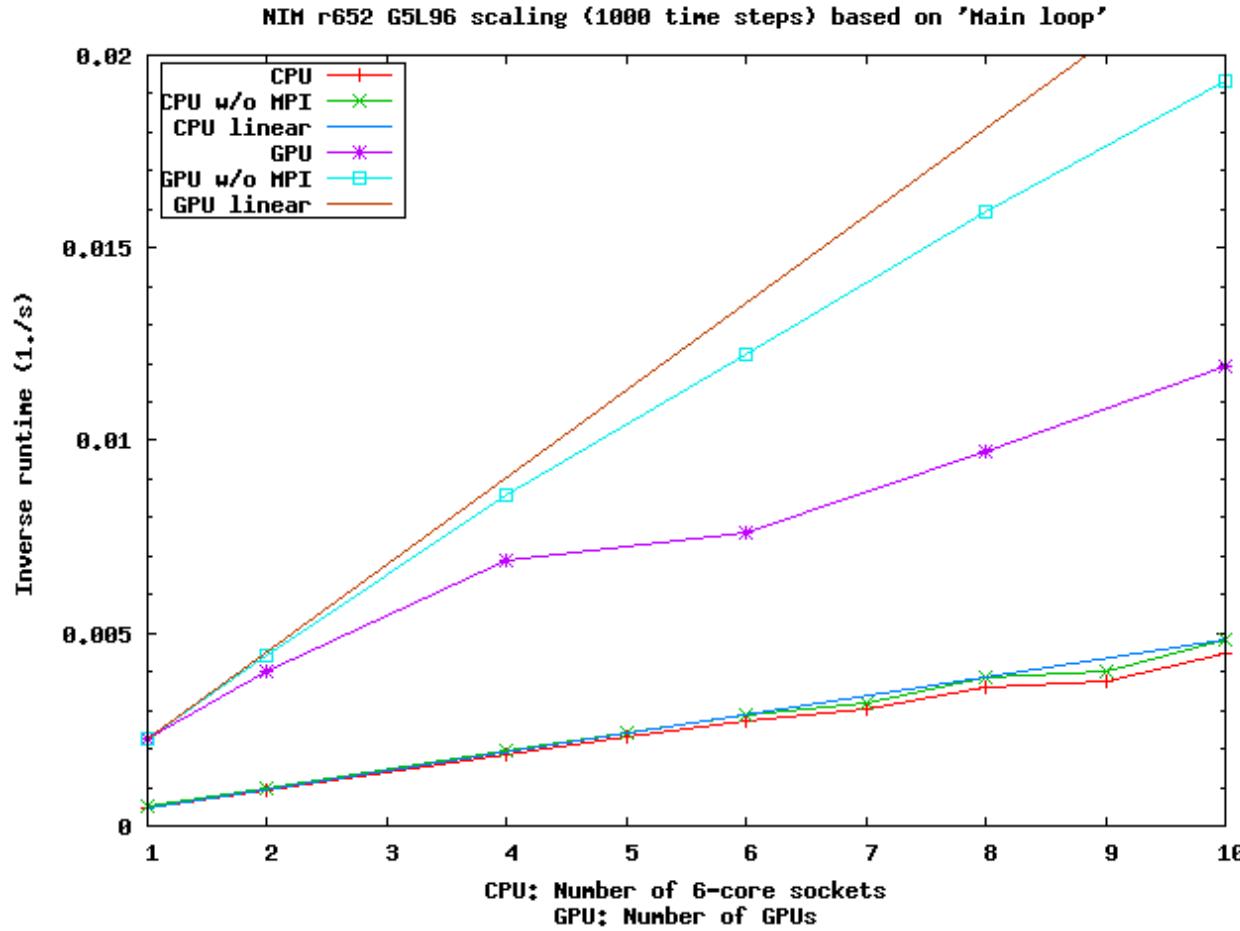
- Used PAPI to count flops (Intel compiler)
 - Requires `-O1` (no vectorization) to be accurate!
 - 2nd run with `-O3` (vectorization) to get wallclock

$$\frac{2.8 * 10^{12}}{940} = 2.98 Gflops/sec$$

27% of peak on Westmere 2.8 GHz!



NIM scaling

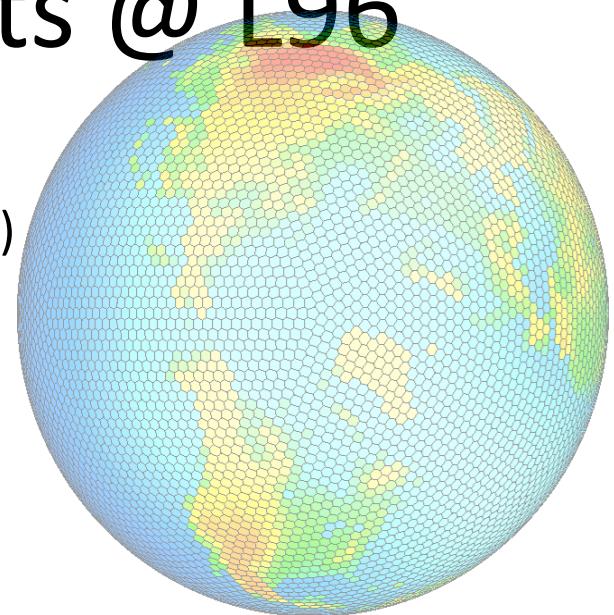


Titan Summit



Memory requirements @ L96

- A doubling of horizontal resolution implies:
 - 4x increase in horizontal points (and memory)
 - 2x increase in model time step
- GPU memory is the limiting factor
- Assume 3GB memory per GPU



	G4	G5	G6	G7	G8	G9	G10	G11
resolution	480KM	240KM	120KM	60KM	30 KM	15 KM	7 KM	3.5 KM
horizontal points	2.5K	10K	40K	160K	640K	2560K	10,000K	40,000K
memory	.25GB	1GB	4GB	16GB	64GB	256GB	1TB	4TB
# GPUs	1	1	2	6	22	86	342	1366



Current Software Status

- Full dynamics runs on CPU or GPU
 - ~5X speedup socket to socket on GPU
 - GPU solution verified correct by comparing output field diffs vs. applying rounding-level perturbation
- Dummy physics can run on CPU or GPU
- Single-source
 - GPU directives ignored in CPU mode
- NO constructs that look like:

```
#ifdef GPU  
<do this>  
#else  
<do that>  
#endif
```



Next Software Tasks

- Optimize communication
 - Asynchronous send/recv (compute perimeter first)
 - Cut down number of synch points
- Bring HMPP and PGI directive-based implementations up to date
- Bring software branch up to date with science trunk (enable running real physics)
- GPU-ize real physics

